Big Data Science Needs Big Data Middleware

Bill Howe
University of Washington
billhowe@cs.washington.edu

There has been a "Cambrian explosion" of big data systems proposed and evaluated in the last eight years, but relatively little understanding of how these systems or the ideas they represent compare and complement one another. In enterprise and science situations, "one size is unlikely to fit all": we see analytics teams running multiple systems simultaneously. However, the highest level of abstraction for interoperability achieved in practice is basically at the file system; for example, HDFS. At the same time, there has been some convergence around higher-level data models (relations, arrays, graphs) and higher-level computational models (relational algebra, parallel data-flow, iteration, linear algebra).

As a result, the design space seems narrower than the implementation space, suggesting an opportunity to build a common "complexity hiding" interface to all these seemingly disparate systems to make them easier to compare, easier to use together, and perhaps to improve overall performance by affording cross-platform, federated optimization.

We are exploring a common programming model for big data systems, subscribing to three design principles:

- *Algebra at the core.* We are less interested in ad hoc engineering solutions that bridge various systems than in identifying and surfacing the primitives, operators, algorithms, and optimization opportunities they share. At the same time, we want to avoid "regressing to the mean" and destroying any competitive advantages or unique capabilities each system may offer.
- *Parallel at the core.* We are less interested in general purpose (serial) programming than in "intrinsically parallel" abstractions. For example, calling out to R running on a single machine in main memory as an intermediate step is not going to work.
- *Iteration at the core.* We need to support multi-pass algorithms to express analytics tasks; first-order queries aren't enough.

Motivated by these ideas, we are working to expand the University of Washington Myria system to act a comprehensive shared interface for big data systems, regardless of model, system, or task.

Myria is a hosted Big Data management and analytics service consisting of three components:

- MyriaX: A big-data execution engine emphasizing asynchronous iterative processing.
- MyriaQ: A language translation layer, shared optimizer, and query execution coordinator supporting a) multiple input languages and b) multiple back-end systems, of which MyriaX is one. We emphasize a new imperative, iterative, language called MyriaL.
- MyriaWeb: A web-based editor and IDE designed for direct use by analysts, with support for collaborative editing through a shared workspace, visual performance analysis and debugging, and interactive algorithm.

To register a system with MyriaQ, the sytem designer provides four components by extending appropriate classes in the MyriaQ Python library:

- An AST for the input language, API, or algebra.
- A mapping from the MyriaQ algebra into the AST.
- (optional) A set of custom optimization rules required to generate appropriate plans; we do not discourage these rules from calling specialized algorithms or UDFs to optimize particular cases. However, if the compilation process essentially degenerates into a lookup table mapping every possible input to some special-case algorithm, then that is something we want to learn as part of this research. Indeed, the number of rules required to support a suite of benchmark queries may be a good quantitative metric for ease of use.
- (optional) An implementation of an Administrative API that includes methods for query execution, schema browsing, monitoring and logging, fetching data, killing queries, restarting the system, collecting statistics, and managing users. Not all of these methods need be implemented for basic operation, but they provide a richer experience for the end user.

Data scientists need to be insulated from the complexity and uncertainty that is dominating the systems research in big data today; we can't ask them to learn and re-learn a new API every month, along with the algorithmic tricks and configuration practices needed for decent performance. But a shared interface to big data systems will not only make things easier for end users — it is critical to advance the science. To do big data systems research today takes a phenomenal effort: N systems must be installed and maintained, and M applications must be implemented and tuned on each of them. As a result, corners are cut: experiments compare just two systems, focus on only simple, narrow use cases, or both. As a field, we must make it significantly easier to do good science, evaluate realistically complex applications, and compare a variety of state of the art systems. We hypothesize that a middleware layer that provides "write once, run anywhere" capabilities would be a significant step towards solving this problem.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.